The R3 Model: Verilog-A Code

Colin McAndrew

Tempe
USA

NEEDS Workshop
November 18-19, 2014

Initial Comments

- Modeling people generally have device/physics background
  - occasionally design (e.g. EKV, Prof. Tsividis)
- Models are really software, model developers:
  - must understand how models and simulators interact
  - must follow standard (good) software practices
    - style
    - source-code control
    - regression testing (automated QA)
      - develop tests in parallel with model
    - documentation
    - structured release practices
- Historic issue: PhD/papers based on physics, not code

Most Common Errors in Compact Model Codes

- Used to be in hand-generated derivatives
  - problem completely removed by use of Verilog-A
- Now is handling of “book-keeping” things
  - polarity flipping (e.g. PMOS instead of NMOS)
  - terminal swapping (e.g. symmetric drain and source)
  - scale (e.g. mapping of netlisted microns to meters)
  - shrink
  - mutiplicity factor
- Last of these automated in Verilog-A, except for mismatch
- Automated QA can do the rest

Generic Issues with Compact Model Codes

- Hand-coded C used to impose a huge barrier to anyone wanting to develop a compact model
- Verilog-A significantly lowered that barrier
  - but does not come with a “bozo filter” to prevent bad code
- People read Verilog-A LRM then think they are an expert
  - universally don’t read existing best practices
  - universally repeat common mistakes
- Will not cover details here
  - more “don’ts” than “do’s”
  - list of “don’ts” expanding all the time
  - updated best practices document in preparation
    - IEEE EDS CM-TAC

R3 Code is Split Into Four Parts

- Two intended to be generic
  - generalMacrosAndDefines.va
  - junctionMacros.va
- Two specific to r3
  - top-level model code in r3.va
  - detailed core calculations in r3Macros.va
- This was done so
  - core r3 calculations can be used as a building block for other models (e.g. LDMOS)
  - nitty-gritty details of r3 calculations do not “clutter” the main code of r3, to make it more readable

Generic Macros and Quantities

- Use these built-in constants

```
//    `M_E          e
//    `M_LOG2E      log2(e)
//    `M_LOG10E     log10(e)
//    `M_LN2        ln(2)
//    `M_LN10       ln(10)
//    `M_PI         pi
//    `M_TWO_PI     pi*2.0
//    `M_PI_2       pi/2.0
//    `M_PI_4       pi/4.0
//    `M_1_PI       1.0/pi
//    `M_2_PI       2.0/pi
//    `M_2_SQRTPI   2.0/sqrt(pi)
//    `M_SQRT2      sqrt(2.0)
//    `M_SQRT1_2    1.0/sqrt(2.0)=sqrt(0.5)
//    `P_CELSIUS0   273.15
//    `P_EPS0       8.854187817e-12
```

Generic Macros and Quantities (2)

- Do **not** use other physical constants, they change over time

```
`define QQ_NIST2004        1.60217653e-19          // (NIST2004)
`define KB_NIST2004        1.3806505e-23           // (NIST2004)
`define EPS_OX             3.45313324863e-11       // `P_EPS0*3.90 (F/m)
`define EPS_SI             1.035939974589e-10      // `P_EPS0*11.7 (F/m)
`define oneSixth           0.1666666666666667
`define oneThird           0.3333333333333333
`define twoThirds          0.6666666666666667
```

- Compiler should optimize (pre-compute) things like 1.0/6.0
  - but don't bank on it
- Include > 16 digits
  - why?

Generic Macros and Quantities (3)

- Clipping/clamping macros
  - to enforce parameter ranges
  - to limit values for voltages and temperature
    - to prevent numerical evaluation problems, help convergence
    - clipping done with an “if” condition so not C_∞ smooth
      - computationally efficient, does not “warp” behavior in normal range
    - clipping ranges of 0.1 and 1.0 for voltages, temperature
  - smooth and hard clamping provided
- Standard parameter declaration types
  - documentation automatically generated from these
- Macros for simulation variables often used in models

Generic Macros and Quantities (4)

- Linearized exponential
  - to prevent numerical computation issues
  - physically something else will always limit currents/charges
    - e.g. parasitic series resistance
- collapsibleR macro
  - if a resistor value is changed from a "normal" value to zero (or a very small value), the formulation should be changed from a current contribution to a voltage contribution
  - implicitly adds an extra system variable (the current)
  - dynamically changing formulation is tricky, not standardized
  - hopefully this may become standard, but could change

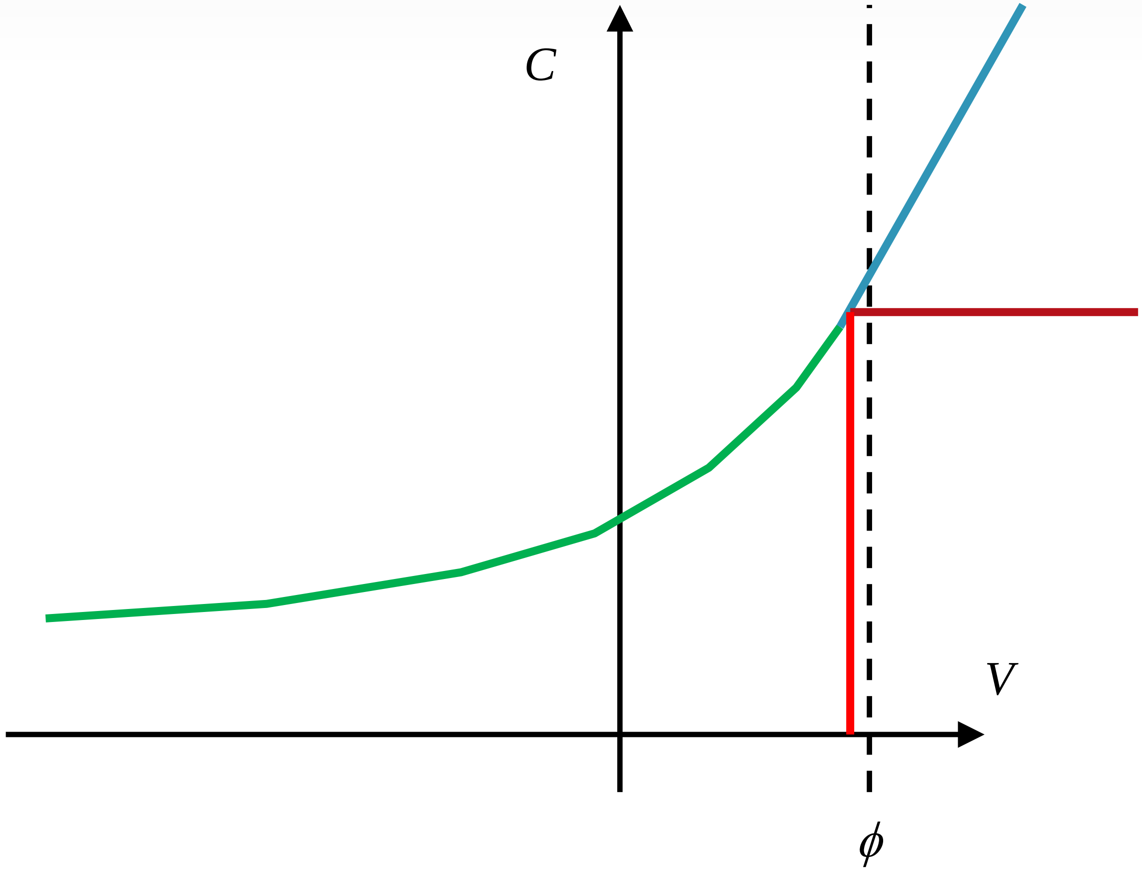
Junction Macros

- *pn*-junctions found in many semiconductor devices
  - core (e.g. diodes, BJTs)
  - parasitic (e.g. MOS transistors)
- No point in every compact model re-inventing the wheel
  - these macros intended to help avoid that
- Junction depletion charge at forward bias
  - what happens when V approaches or exceeds ϕ?

$$C_{depl} = \frac{C_0}{\left(1 - \frac{V}{\phi}\right)^m}$$

freescale™

Junction Macros (2)



Junction Macros (3)

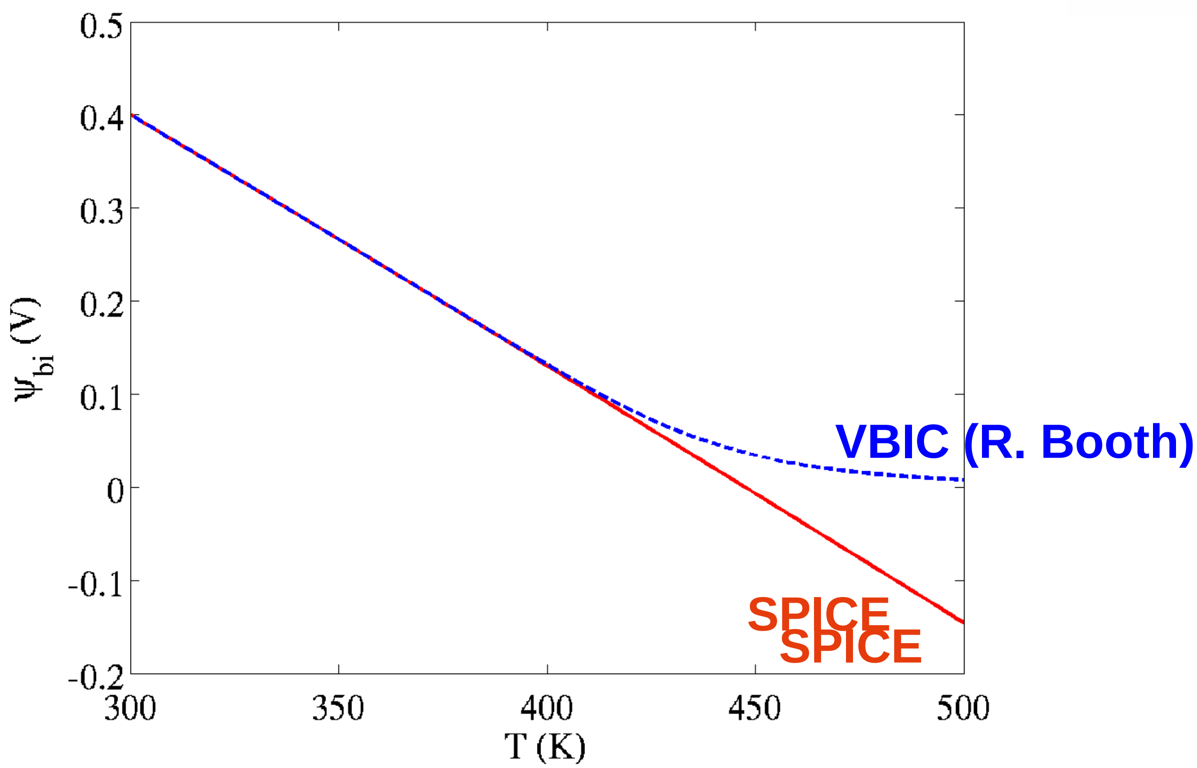
- Linear extrapolation is general SPICE approach
- Flattening off is used in some models
- Physically what happens is C_{depl} peaks and then drops smoothly to zero
  - but this can cause convergence problems
- Gets overwhelmed by diffusion charge (exponential in V) so doesn't really matter
- Junction macros include all three approaches
  - with selectable hard or smooth transition
  - with selectable non-zero or zero value for $V=0$
  - latter is more computationally expensive, but needed for Early effect modeling for BJTs

Junction Macros (4)

- Also includes
  - junction area/perimeter component current and charge models
  - junction breakdown model
  - proper junction shot noise model
  - asymptotically correct junction built-in potential temperature mapping

Junction Macros (4)

$$\phi_{bi} = \frac{kT}{q}\ln\left(\frac{N_A N_D}{n_i^2(T)}\right)$$



r3.va

- Let’s go through the code now

Macro for core R3 Calculation

- Basically separates all of the tricky and complex code for model evaluation
- Much of it is one-to-one with expressions derived in the “physics of r3” slides
  - part of homework assignment is to identify those
  - two most complex parts are handling pinch-off and computing the saturation voltage
  - do **not** worry about details of pinch-off computations
    - only relevant for JFETs

V_{sat} Calculation

- Complex part of the model
  - transitions to drain pinch-off and full pinch-off must be smooth
  - smoothing/interpolation function used maintains symmetry

$$V_{sat} \text{ is solution of } \frac{\partial}{\partial V}\left(V \frac{1 - d_f \sqrt{d_{pe} + V}}{1 + \frac{\frac{V}{L} - E_{co}}{E_{cr}}} \right) = 0 \quad \text{where} \quad d_{pe} = d_p - 2V_{c1}, \quad L_{DE} = L\left(E_{cr} - E_{co}\right)$$

$$\frac{d_f^2}{4L_{DE}^2} V_{sat}^4 + \frac{3d_f^2}{2L_{DE}} V_{sat}^3 + d_f^2\left(\frac{9}{4} + \frac{p_e}{L_{DE}}\right) V_{sat}^2 + \left(3d_f^2 p_e - 1\right) V_{sat} + p_e\left(d_f^2 p_e - 1\right) = 0$$

freescale™

V_{sat} Calculation (

$$\frac{d_f^2}{4L_{DE}^2}V_{sat}^4 + \frac{3d_f^2}{2L_{DE}}V_{sat}^3 + d$$

A quartic equation,

$$x^4 + ax^3 + bx^2 + cx + d = 0,$$

has the *resolvent cubic equation*

$$y^3 - by^2 + (ac - 4d)y - a^2d + 4bd - c^2 = 0.$$

A cubic equation $y^3 + py^2 + qy + r = 0$ may be reduced to the form,—

$$x^3 + ax + b = 0$$

by substituting for y the value, $x - \frac{p}{3}$. Here

$$a = \tfrac{1}{3}(3q - p^2) \text{ and } b = \tfrac{1}{27}(2p^3 - 9pq + 27r).$$

For solution let,—

$$A = \sqrt[3]{-\frac{b}{2} + \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}}, \quad B = \sqrt[3]{-\frac{b}{2} - \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}},$$

then the values of x will be given by,

$$x = A + B, \quad -\frac{A+B}{2} + \frac{A-B}{2}\sqrt{-3}, \quad -\frac{A+B}{2} - \frac{A-B}{2}\sqrt{-3}.$$

```
if (iecrit>0.0) begin \
      a0       =  dfsq*dpe*d
      a1       = -1.0+3.0*df
      a2       =  dfsq*(9.0/
      a3       =  1.5*dfsq/l
      a4       =  4.0*lde*ld
      dvar     =  a0*a4; \
      cvar     =  a1*a4; \
      bvar     =  a2*a4; \
      avar     =  a3*a4; \
      asq      =  avar*avar; \
      pvar     = -bvar; \
      qvar     =  avar*cvar-4.0*dvar; \
      rvar     =  4.0*bvar*dvar-cvar*cvar-dvar*asq; \
      aa       =  qvar-pvar*pvar*`oneThird; \
      bb       =  rvar-pvar*(qvar+2.0*aa)/9.0; \
      aa3d27   =  aa*aa*aa/27.0; \
      dd       =  0.25*bb*bb+aa3d27; \
      sd       =  sqrt(dd); \
```

QA

- Simple QA test specification, and code to run simulations, is provided with r3
- This is CMC QA code